AMENDMENTS TO THE SPECIFICATION

Amend the title to -- METHOD AND APPARATUS FOR REDUCING LOGIC ACTIVITY IN A MICROPROCESSOR USING REDUCED BIT WIDTH SLICES THAT ARE ENABLED OR DISABLED DEPENDING ON OPERATION WIDTH--.

Amend the paragraph starting at page 4, line 10, with the following replacement paragraph.

Figure 2 is a block diagram of a first embodiment of a microprocessor pipeline structure of the present invention having an instruction cache 201 and instruction buffer 202 similar to 101 and 102 as explained with references to Figure 1, and incorporating width determination (WD) Logic 220 and bit width slices consisting of 4 slices 230, wherein a slice 230 is of 8-bit granularity. Each slice 230 consists of a portion of the register file RF 250, a portion of the ALU 260, and a portion of the cache memory 270. In Figure 2, the lowermost or least significant slice is illustrated on the right, and the uppermost or most significant slice is illustrated on the left, with a data carry operation from a lesser significant slice to a more significant slice. The concatenation of these 4 slices 230 creates a full-width processor, as required by the processor architecture.

Amend the paragraph starting at page 6, line 9, with the following replacement paragraph.

From the foregoing definition of the effective operation width, Figure 4 illustrates graphs 410, 420 on the potential for varying bit-width computing, and illustrates the average operation width distributions of the SPECInt2000 and Mediabench. These graphs show that the average embedded application instruction is skewed in the direction of narrow width words, creating the impetus for narrow bit-width computing pursuant to the present invention.

Amend the paragraph starting at page 8, line 15, with the following replacement paragraph.

Figure 5 illustrates a second embodiment of the present invention having a cache tags 580 and width tags 581 similar to 280 and 281 as explained with reference to Figure

2, and also having a varying bit width architecture, and shows a pipeline microprocessor structure consisting of 3 slices, where 2 slices 540 are of width 8-bits and 1 slice 530 is of width 16-bits. The embodiment here illustrates that an implementation can be of varying number of slices where the slices can be of varying bit widths as well. Again, each slice consists of a portion of the register file RF 550, the ALU 560, and a portion of the cache memory 570.

Amend the paragraph starting at page 9, line 24, with the following replacement paragraph.

Referring to Figures 6, 7 and 8, a lookup of the RFtags table is performed for the 2 source operands Src1, Src2 and the result register Dst, as shown at 710 in Figure 7. Considering the operation of the instruction, the bitmask information about the value widths of the 2 source operands (Src1, Src2) are compared at 610, Figure 6, and in the WD logic, as shown at 730, Figure 7. Referring to Figure 6 following start 600, the widths Src1 and Src2 are compared at 610, and, the narrower width of Src1 and Src2 is selected as the interim EW (effective operation width) at 620. Then the kind of operation is checked at 630. If the operation is a logic operation at 630, then the interim EW is the final EW at 680. This is because a logic operation, such as an AND or OR operation, by its very nature, does not result in an overflow. Whereas, if the operation is an arithmetic operation, such as an addition, then there is the possibility that an overflow can occur. In that case the next step is to check for an overflow at 640 and 650, Figure 6, and at 820, Figure 8.

Amend the paragraph starting at page 10, line 7, with the following replacement paragraph.

The leading 2 bits of the narrower operand are compared against the matching bits of the other operand at 640. In the case where both operands are of the same width, the leading 2 bits of both operands are compared against each other. If it is determined that overflow is not possible, then the interim EW is the final EW at 680. However, if there is an overflow, then EW is EW + 1 at 660. In the case where EW is 2 (for a 16 bit computation), for the embodiment of Figure 5, this addition should take EW to a 4 (for a 32 bit computation) since the implementation of Figure 5 skips a 24-bit computation).

The new EW is checked to determine if it is less than or equal to 2 (i.e. 2 bytes) at 670. If the new EW turns out to be less than or equal to 2, then return to 640, where the leading 2 bits of the EW "operand" is again compared against the matching 2 bits of the other operand for possible overflow at 650. However, if the EW is larger than 2 at 670, then the final EW is the new EW. The final EW is next compared against the width of the result register Dst at 690. If EW is less than the width of Dst at 690, then quasi-simple work is performed at 691 to fill or reset the upper portion of Dst. No quasi-simple work otherwise, as indicated at 692.

Amend the paragraph starting at page 13, line 17, with the following replacement paragraph.

Figure 13 is a logic flow diagram that illustrates, following a determination of ~~how to determine~~ the width of an operand or a result value at 130, a comparison at 131 determines if the result = X"0000_0000, and if yes, then at 132, the result-width=0 bit. A comparison at 133 determines if the result (bit 31 to bit 8) = (X"0000_00 or X"Ffff_Ff), and if yes, then at 134 the result-width = 8 bits. A comparison at 135 determines if the result (bit 31 to bit 16) = (X"0000 or X"Ffff), and if yes, then at 136 the result-width=16 bits. A comparison at 137 determines if the result (bit 31 to bit 24) = (X"00 or X"Ff), and if yes, then at 138 the result – width = 24 bits, and if not then at 139 the result-width = 32 bits. This illustration is shown for up to 32-bit wide values. Even though the comparisons 131, 133, 135, and 137 are illustrated serially in Figure 13, they can also be carried out in parallel.